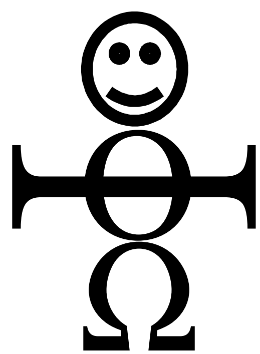
CSE 548: Analysis of Algorithms

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2014



Asymptotic Stickman
(by Aleksandra Patrzalek, SUNY Buffalo)

Some Mostly Useless Information

- **Lecture Time:** MoWe 2:30 pm - 3:50 pm
- **Location:** Humanities 1003, West Campus
- **Instructor:** Rezaul A. Chowdhury
- **Office Hours:** MoWe 11:30 am - 1:00 pm, 1421 Computer Science
- **Email:** rezaul@cs.stonybrook.edu
- **TA:** TBA
- **Class Webpage:**

  http://www.cs.sunysb.edu/~rezaul/CSE548-S14.html

Prerequisites

- **Required:** Some background (undergrad level) in the design and analysis of algorithms and data structures
    - fundamental data structures (e.g., lists, stacks, queues and arrays)
    - discrete mathematical structures (e.g., graphs, trees, and their adjacency lists & adjacency matrix representations)
    - fundamental programming techniques (e.g., recursion, divide-and-conquer, and dynamic programming)
    - basic sorting and searching algorithms
    - fundamentals of asymptotic analysis (e.g., $O(\cdot)$, $\Omega(\cdot)$ and $\Theta(\cdot)$ notations)
- **Required:** Some background in programming languages (C / C++)

Topics to be Covered

The following topics will be covered (hopefully)

- recurrence relations and divide-and-conquer algorithms
- dynamic programming
- graph algorithms (e.g., network flow)
- amortized analysis
- advanced data structures (e.g., Fibonacci heaps)
- cache-efficient and external-memory algorithms
- high probability bounds and randomized algorithms
- parallel algorithms and multithreaded computations
- NP-completeness and approximation algorithms
- the alpha technique (e.g., disjoint sets, partial sums)
- FFT (Fast Fourier Transforms)

Grading Policy

- Four Homework Problem Sets
  (highest score 15%, lowest score 5%, and others 10% each): 40%
- Two Exams (higher one 30%, lower one 15%): 45%
  - Midterm (in-class): Mar 12
  - Final (in-class): May 7
- Scribe note (one lecture): 10%
- Class participation & attendance: 5%

Textbooks

Required

- Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein.
  Introduction to Algorithms (3rd Edition), MIT Press, 2009.

Recommended

- Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani.
  Algorithms (1st Edition), McGraw-Hill, 2006.
- Jon Kleinberg and Éva Tardos.
  Algorithm Design (1st Edition), Addison Wesley, 2005.
- Rajeev Motwani and Prabhakar Raghavan.
  Randomized Algorithms (1st Edition), Cambridge University Press, 1995.
- Vijay Vazirani.
  Approximation Algorithms, Springer, 2010.
- Joseph JáJá.
  An Introduction to Parallel Algorithms (1st Edition), Addison Wesley, 1992.

What is an Algorithm?

An algorithm is a ***well-defined computational procedure*** that solves a well-specified computational problem.

It accepts a value or set of values as ***input***, and produces a value or set of values as ***output***

Example: ***mergesort*** solves the ***sorting problem*** specified as a relationship between the input and the output as follows.

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Desirable Properties of an Algorithm

√ Correctness

– Designing an incorrect algorithm is straight-forward

√ Efficiency

– Efficiency is easily achievable if we give up on correctness

Surprisingly, sometimes incorrect algorithms can also be useful!

– If you can control the error rate

– Tradeoff between correctness and efficiency:

Randomized algorithms

(Monte Carlo: always efficient but sometimes incorrect,

Las Vegas: always correct but sometimes inefficient)

Approximation algorithms

(always incorrect!)

How Do You Measure Efficiency?

We often want algorithms that can use the available resources efficiently.

Some measures of efficiency

- time complexity
- space complexity
- cache complexity
- I/O complexity
- energy usage
- number of processors/cores used
- network bandwidth

Goal of Algorithm Analysis

Goal is to predict the behavior of an algorithm without implementing it on a real machine.

But predicting the exact behavior is not always possible as there are too many influencing factors.

Runtime on a serial machine is the most commonly used measure.

We need to model the machine first in order to analyze runtimes.

But an exact model will make the analysis too complicated!
So we use an approximate model (e.g., assume unit-cost Random Access Machine model or RAM model).

We may need to approximate even further: e.g., for a sorting algorithm we may count the comparison operations only.

So the predicted running time will only be an approximation!

Performance Bounds

- **worst-case complexity:** maximum complexity over all inputs of a given size
- **average complexity:** average complexity over all inputs of a given size
- **amortized complexity:** worst-case bound on a sequence of operations
- **expected complexity:** for algorithms that make random choices during execution (randomized algorithms)
- **high-probability bound:** when the probability that the complexity holds is $\geq 1 - \frac{c}{n^{\alpha}}$ for input size n, positive constant c and some constant $\alpha \geq 1$

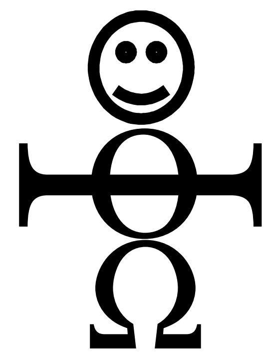
Asymptotic Bounds

We compute performance bounds as functions of input size n.

Asymptotic bounds are obtained when $n \to \infty$.

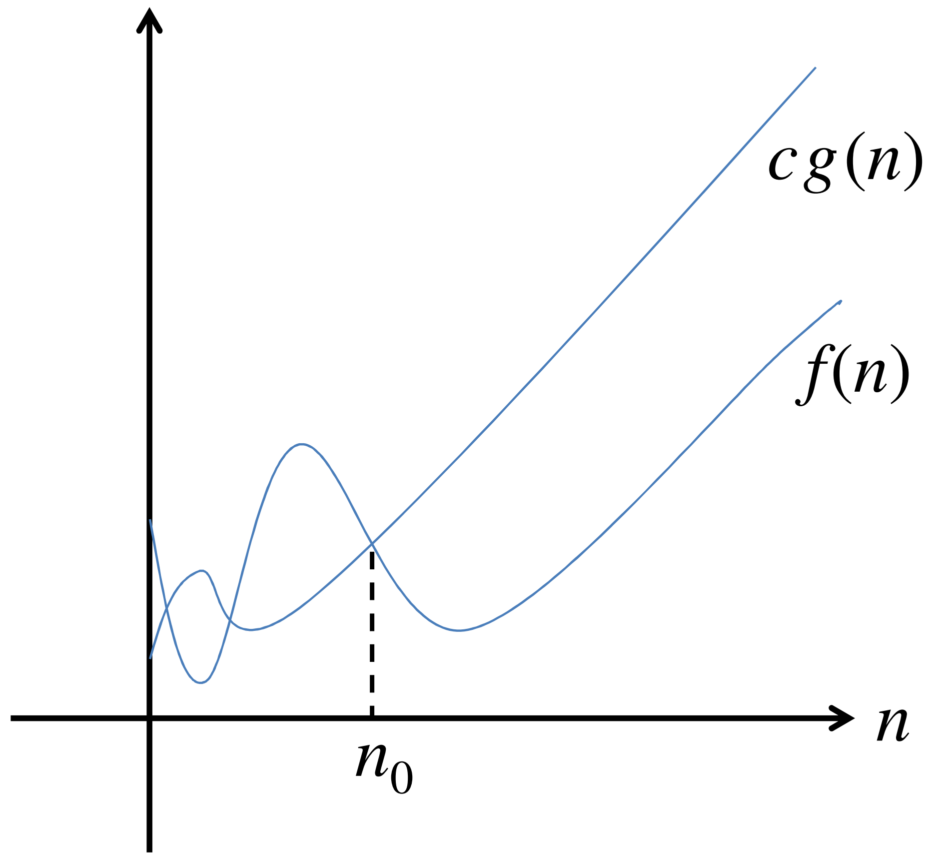
Several types of asymptotic bounds

- upper bound (O-notation)
- strict upper bound (o-notation)
- lower bound (Ω-notation)
- strict lower bound (ω-notation)
- tight bound (Θ-notation)



Asymptotic Stickman
(by Aleksandra Patrzalek)

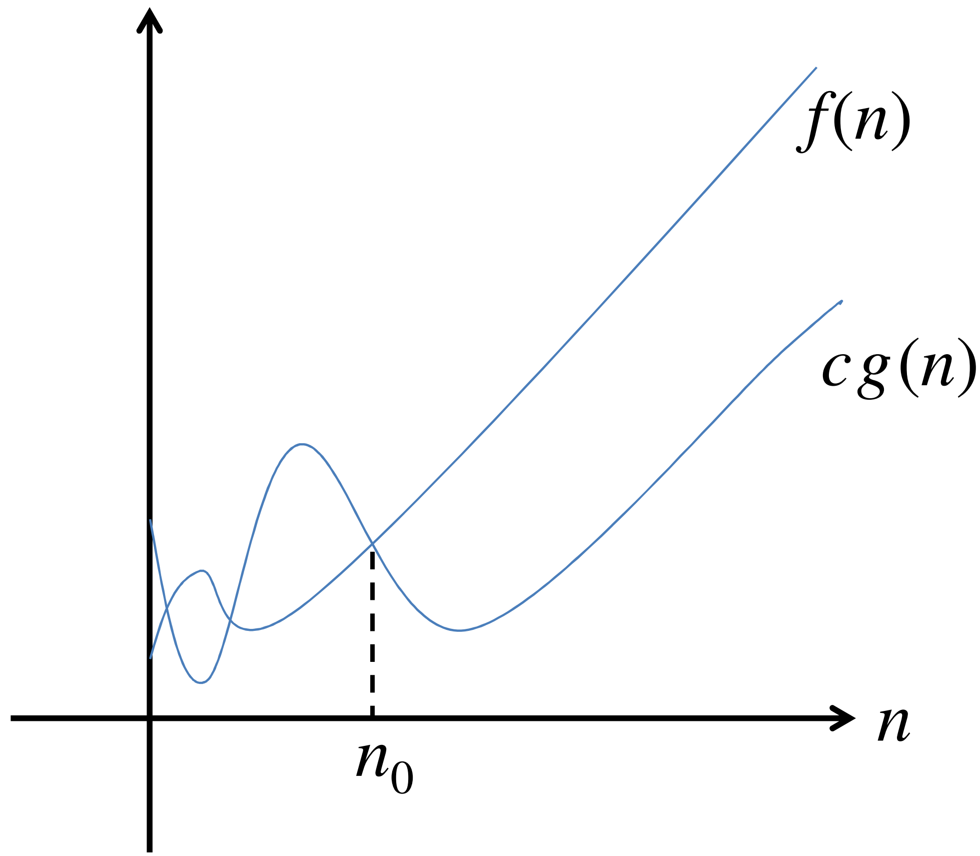
Asymptotic Upper Bound (O-notation)



$$O(g(n)) = \left\{ \begin{matrix} f(n): there\ exist\ positive\ constants\ c\ and\ n_0\ such\ that \\ 0 \le f(n) \le cg(n)\ for\ all\ n \ge n_0 \end{matrix} \right\}$$

$$O(g(n)) = \left\{ \begin{matrix} f(n): there\ exists\ a\ positive\ constant\ c\ such\ that \\ \lim_{n\to\infty} \left(\dfrac{f(n)}{g(n)} \right) \le c \end{matrix} \right\}$$

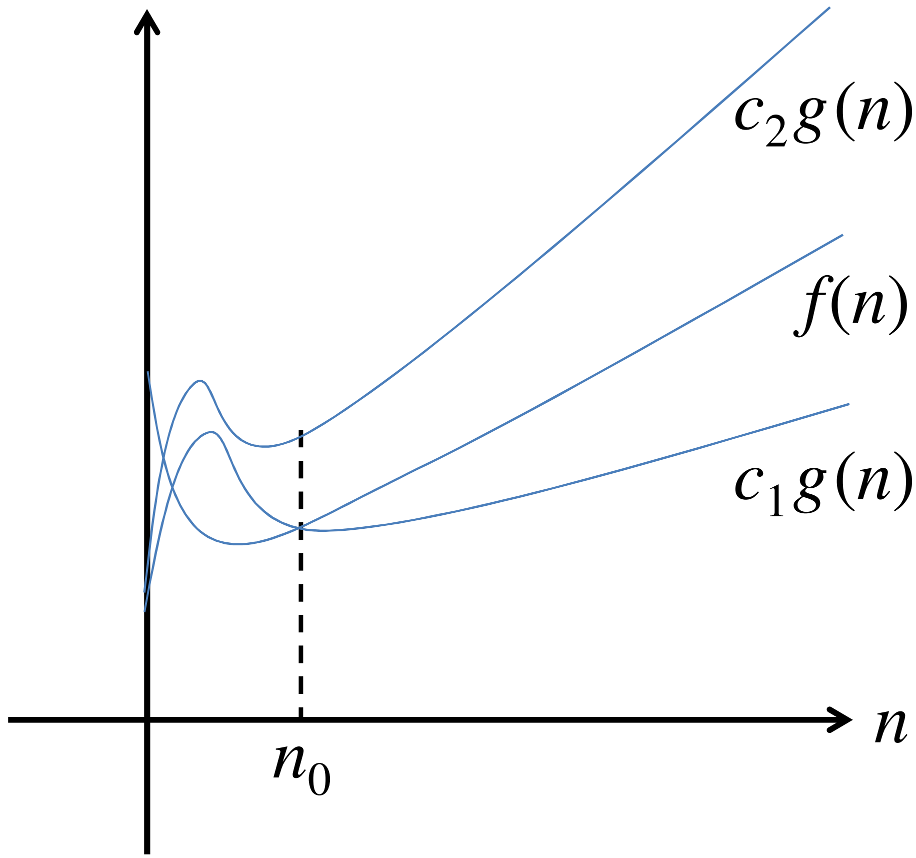
<u>Asymptotic Lower Bound (Ω-notation)</u>



$$\Omega(g(n)) = \left\{ \begin{array}{c} f(n): there\ exist\ positive\ constants\ c\ and\ n_0\ such\ that \\ 0 \le cg(n) \le f(n)\ for\ all\ n \ge n_0 \end{array} \right\}$$

$$\Omega(g(n)) = \left\{ \begin{array}{c} f(n): there\ exists\ a\ positive\ constant\ c\ such\ that \\ \lim_{n \to \infty} \left(\frac{f(n)}{g(n)} \right) \ge c \end{array} \right\}$$

Asymptotic Tight Bound (Θ-notation)



$$\Theta(g(n)) = \left\{ \begin{array}{c} f(n)\text{: there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that} \\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

$$\Theta(g(n)) = \left\{ \begin{array}{c} f(n)\text{: there exist positive constants } c_1 \text{ and } c_2 \text{ such that} \\ c_1 \le \lim_{n\to\infty} \left(\dfrac{f(n)}{g(n)} \right) \le c_2 \end{array} \right\}$$

Asymptotic Strict Upper Bound (o-notation)

$$O(g(n)) = \left\{ \begin{matrix} f(n): there\ exist\ positive\ constants\ c\ and\ n_0\ such\ that \\ 0 \le f(n) \le cg(n)\ for\ all\ n \ge n_0 \end{matrix} \right\}$$

$$O(g(n)) = \left\{ \begin{matrix} f(n): there\ exists\ a\ positive\ constant\ c\ such\ that \\ \lim_{n\to\infty} \left(\frac{f(n)}{g(n)} \right) \le c \end{matrix} \right\}$$

$$o(g(n)) = \left\{ f(n): \lim_{n\to\infty} \left(\frac{f(n)}{g(n)} \right) = 0 \right\}$$

Asymptotic Strict Lower Bound (ω-notation)

$$\Omega\big(g(n)\big) = \left\{ \begin{matrix} f(n)\text{: } \textit{there exist positive constants } c \textit{ and } n_0 \textit{ such that} \\ 0 \le cg(n) \le f(n) \textit{ for all } n \ge n_0 \end{matrix} \right\}$$

$$\Omega\big(g(n)\big) = \left\{ \begin{matrix} f(n)\text{: } \textit{there exists a positive constant } c \textit{ such that} \\ \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) \ge c \end{matrix} \right\}$$

$$\omega\big(g(n)\big) = \left\{ f(n)\text{: } \lim_{n\to\infty}\left(\frac{g(n)}{f(n)}\right) = 0 \right\}$$